CRITICAL I/O WHITE PAPER

XGE 10Gb Ethernet Software Support Overview

Linux 2.6.x, 3.x
VxWorks 6.x
Windows

Abstract

The XGE 10G series of 10 GbE NICs provides dual port 10 Gigabit Ethernet (10GbE) connectivity for embedded systems with the high performance characteristics that are essential for data intensive real-time systems, yet maintaining 100% interoperability and compatibility with standard or enhanced Ethernet infrastructures.

TCP and UDP send/receive performance of up to 800 MB/s under VxWorks and 1200 MB/s under Linux and Windows can be achieved using XGE 10G drivers. The standard sockets API, a high performance streaming API, and a low latency RDMA API are supported.

This paper describes the XGE 10G drivers available for Linux, VxWorks, and Windows.

Revision 4/1/2014

CRITICAL IO

Critical I/O XGE 10 Gb Ethernet Software Support

The XGE 10G series of 10 GbE NICs provide dual port 10 Gigabit Ethernet (10GbE) connectivity to embedded systems with the high performance characteristics that are essential for data intensive real-time systems, yet maintaining full interoperability and compatibility with standard or enhanced Ethernet infrastructures.

The XGE 10G NICs provide a balanced architecture which offers hardware acceleration for large data transfers with the flexibility of a programmable protocol processor to significantly improve network performance. These hardware offloads reduce the CPU cycles and burden required for 10 GbE networking, maximizing usable network bandwidth without sacrificing host CPU processing capability.

XGE 10G Driver Models

The Critical I/O XGE Linux, VxWorks and Windows Drivers allow user access to the 10GbE network interface through three different methods:

- A standard operating system sockets API
- A special high performance UDP Stream API
- A Remote Direct Memory Access (RDMA) API

Network Driver -- Sockets API Model

The standard network socket API model connects the XGE driver through the operating system sockets interface and network stack. This allows new and existing user developed socket applications and standard network applications like FTP, Telnet, NFS etc. to make use of the XGE interface. Network performance and CPU loading is very good, but loading and transfer latencies may be somewhat higher (as compared to the stream or RDMA models) due to the interaction of the XGE 10G hardware with the operating system network stack.

UDP Stream Driver - Stream API Model

UDP Streaming provides a high performance offload data transfer model which bypasses the normal operating system network stack to more fully leverage the offload capabilities of the XGE 10G hardware. As the standard BSD socket datagram send/receive API is very limited, access to the UDP streaming functionality is provided via a specialized UDP streaming send/receive API. This specialized API provides very high performance UDP sends and receives with low host CPU loading. With a typical i7 CPU hosting the CIO UDP Stream driver, full 10GbE line rate sends and receives can be achieved using less a 5% loading of one CPU core.

Note that the “on the wire” network traffic for the UDP stream mode is a series of completely standard UDP datagrams. Thus no special network hardware or remote NIC hardware/software is needed.

A “stream” is defined as a flow of data (a series of UDP datagrams) transferred between two UDP “endpoints”, where each endpoint is defined by IP address and UDP port. For example, a connection between [192.168.5.1: 1005] and [192.168.5:1025] is a stream. (IP addresses and UDP ports can also be wildcards).

The hardware and firmware inside the XGE NIC have the ability to "steer" incoming UDP traffic to a specific set of receive buffers associated only with that specific stream, and also have the ability to strip off the various Ethernet/IP/UDP packet headers prior to writing the payload data into those buffers. The set of receive buffers can be located anywhere in PCIe address space, such as within an external GPU or FPGA card memory. This is significant because data will be deposited by the NIC hardware right where it is needed, with no data copies and no network stack overhead.

For sends, the UDP streaming interface is used to send application supplied blocks of data as a stream of UDP datagrams, with the datagram size being a user specified value. Datagrams must be sized to fit within the current Ethernet frame size. The XGE offload hardware will break the application supplied blocks of data into a sequence of UDP datagrams, which relieves the host processor from the overhead of doing multiple individual datagram sends. Thus the application may pass very large blocks of data to the UDP streaming API to be sent, with no CPU involvement needed to perform that datagram sends, other that the initial send setup.

For receives, the application provides a large data buffer to the UDP streaming API that is to be filled with received datagrams for a defined IP/port. The offload hardware will fill the buffer with received datagrams. The application may pass very large receive buffers to the UDP streaming API to be filled, with negligible CPU involvement required to fill the buffers with data after the initial receive setup. The data stream is delivered to the application via a series of UDP datagrams that are written directly into application data buffers after stripping off the datagram header information.

Source and destination buffer addresses are completely arbitrary raw PCIe addresses. For example, one can define a 1 MB buffer located within a PCIe accessible external GPU's memory. A single call to the CIO driver is made to associate this "big" buffer with a specific stream. The driver and NIC firmware will divide the 1MB buffer into individual datagram buffers that are right size to receive the payload portion of the incoming datagram stream. Only when the 1MB buffer is completely filled with incoming datagrams will the NIC generate an interrupt which will then result in the driver providing a user RX completion notification. Only a single driver call is needed to set up to receive the 1MB of data directly into GPU memory, and then provide the completion to the user application when all of the data has been received. Furthermore, all packet headers are stripped off which results in the payload data being packed contiguously into memory.

RDMA Driver – RDMA/IB Verbs API Model

Remote Direct Memory Access (RDMA) is a capability that allows processor to processor data movement directly between application buffers with minimal CPU involvement. The RDMA API provides functionality to define local RDMA buffer regions which remote RDMA nodes can read and write without local CPU involvement, either directly or using RDMA based messaging.

RDMA is an extremely fast and efficient way to move data over a 10 GbE network. Since there is no CPU involvement in the actual movement of data over the network, the effective CPU utilization is close to zero for sufficiently large transfers, and latencies of just a few microseconds are achievable for small transfers.

RDMA over Converged Ethernet (RoCE) is a specific implementation of RDMA that is used bye the XGE NIC hardware. RoCE is a link layer protocol implementation; therefore it is not routable, which limits its use to a single Ethernet subnet. For best performance, RoCE should be

used in conjunction with Enhanced Ethernet (also called Data Center Ethernet or DCE) network switches. Enhanced Ethernet switches provide improved flow control mechanisms to avoid packet loss in cases of network congestion.

Note that RoCE is not the same as iWARP, which is another standard for RDMA over Ethernet. The RoCE protocol defines how to perform RDMA over the Ethernet link layer, while the iWARP protocol defines how to perform RDMA over the Transmission Control Protocol (TCP). The RoCE and iWARP protocols are not interoperable.

XGE Driver Operating System Support

The table below shows the general availability and characteristics of the different driver types and the supported operating systems.

	Linux	**VxWorks**	**Windows**	Supports general networking	Provides ultra-low latency	Provides true zero copy DMA	Requires DCE/CEE network	Principal User API
Network Driver	Yes	Yes (1)	Yes (2)	Yes	No	No	No	Sockets
Stream Driver	Yes	Yes (3)	No	Yes	No	Yes	No	Stream
RDMA Driver	Yes	No	Yes	Yes	Yes	Yes	Yes	RDMA/IB Verbs

(1) VxWorks network driver is included with Stream driver
(2) Windows network driver is included with RDMA driver
(3) VxWorks stream driver planned but not yet available for "-G" version XMCs

XGE Linux Driver Support

Linux Network Driver

The XGE network driver mode of operation uses the standard Linux network stack, thus provides full compatibility with all Linux network applications, and any other applications that use the sockets API. The XGE hardware fully leverages the offload capabilities that are supported by Linux. The principal user interface for the network driver is the standards sockets API.

Linux Network Driver Version Support

The XGE Network Driver standard supports many standard distributions of Linux using kernel versions 2.x and 3.x, and is normally supplied as source code.

Note that most recent standard Linux distributions will already include a network driver suitable for use with XGE hardware.

Linux Stream Driver

The Linux XGE Stream Driver supports two distinct modes of operation, which may be used concurrently. The driver supports general networking via standard Linux network sockets API. The XGE hardware fully leverages the offload capabilities that are supported by Linux. The driver also provides a special high performance offload data transfer mode via a UDP Stream API

which leverages additional XGE hardware offload capabilities. In both modes of operation, everything that "goes on the wire" is always standard IP/Ethernet traffic that is fully compatible with standard Ethernet networks and standard Ethernet NICs

Figure 1 shows the two components of the Linux Stream driver: a standard Linux network driver model and a High Performance Offload driver (e.g. UDP Streaming API) model driver for higher performance data transfer modes. These two parts of the XGE Driver coexist, so socket applications using the network stack and applications using the UDP Stream API can concurrently access the XGE network interfaces.

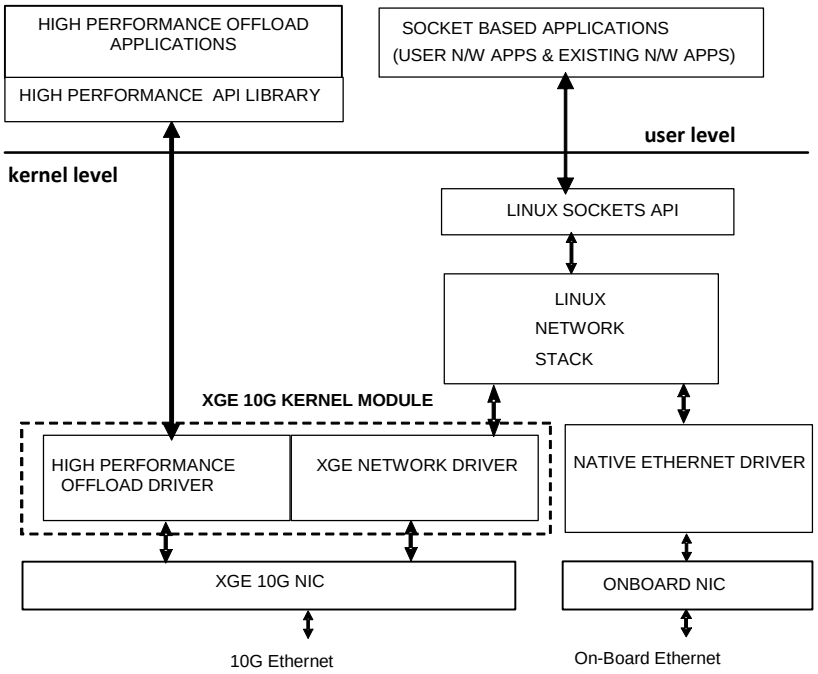

Figure 1. Linux XGE 10G Stream Driver Architecture

Linux UDP Stream API Examples

The UDP Stream API provides the application interface to send and receive streams of UDP datagrams. Examples of the stream functions available within this API are:

xel_init	- Initialize the user level library
xel_end	- Clean up the user level library
xel_udp_smsend_setup	- Set up a socket for UDP stream sends
xel_udp_smsend_multi	- Perform a UDP stream send
xel_udp_smsend_close	- Close a UDP stream send socket
xel_udp_smrecv_setup	- Set up a socket for UDP stream receive
xel_udp_smrecv_multi	- Perform a UDP stream receive
xel_udp_smrecv_close	- Close a UDP stream receive socket

Linux Stream Driver Version Support

The Linux XGE Stream Driver is available for selected Linux 2.6.x kernel and 3.x kernel versions, and may be ported upon request to additional kernel versions.

Linux RDMA Driver

The XGE Linux RDMA driver provides two modes of operation. The network mode uses the standard Linux network stack, thus provides full compatibility with all Linux network applications, and user applications that use the sockets API. The RDMA mode of operation provides for RDMA over Converged Ethernet (RoCE) operation. The principle user APIs for this mode are the IB Verbs API and the RDMA API

Linux RDMA API Examples

The Linux IBV and RDMA APIs are quite rich and a full description is beyond the scope of this document. Shown below are some examples of the user callable RDMA API functions along with brief description of functionality.

rdma_connect	- initiate an active RDMA connection
rdma_listen	- listen for incoming RDMA connections
rdma_accept	- accept a RDMA connection
rdma_disconnect	- disconnect a RDMA connection
rdma_join_multicast	- join a multicast group
rdma_reg_read	- register a local memory buffer for RDMA reads
rdma_reg_write	- register a local memory buffer for RDMA writes
rdma_post_recv	- receive an incoming message sent by a remote peer.
rdma_post_send	- send a message to a remote peer
rdma_post_read	- perform a RDMA read from a remote peer memory buffer.
rdma_post_write	- perform a RDMA write to a remote peer memory buffer.

Linux RDMA Driver Version Support

The Linux XGE RDMA Driver is available for selected Linux 2.6.x kernel and 3.x kernel versions, and may be ported upon request to additional kernel versions

XGE VxWorks Driver Support

VxWorks Network/Stream Driver

The VxWorks XGE Network/Stream Driver supports two distinct modes of operation, which may be used concurrently. The driver supports general networking via standard VxWork network sockets API. The driver also provides a special high performance offload data transfer mode via a UDP Stream API which leverages additional XGE hardware offload capabilities. In both modes of operation, everything that "goes on the wire" is always standard IP/Ethernet traffic that is fully compatible with standard Ethernet networks and standard Ethernet NICs.

Figure 2 shows the two components of the VxWorks Network/Stream driver: a standard VxWorks network driver model, and a High Performance Offload driver (e.g. UDP Streaming) driver model. The network driver mode of operation uses the standard VxWorks network stack, thus provides full compatibility with all VxWorks network applications, and any other applications that use the sockets API. The high performance streaming API mode of operation uses specialized data transfer APIs that allow full use of the offload capabilities of the XGE 10G hardware. Note that the high performance offload mode of operation is only compatible with

VxWorks kernel applications; it is not available for VxWorks Real Time Process (RTP) applications

Two sub-modes of operation are also available using the standard network driver model. The *Moderated* mode provides very good network performance combined with lowest CPU loading through the use of interrupt moderation and coalescing techniques. The tradeoff is slightly lower peak performance, and slightly higher transfer latencies. The *Non-Moderated* mode focuses on achieving the highest possible data rates and the lowest possible transfer latencies, but at the expense of higher CPU loading

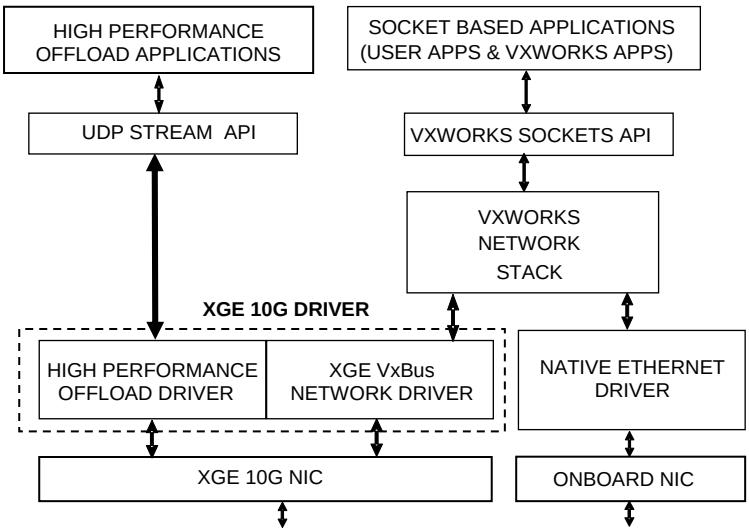

Figure 2. VxWorks XGE 10G Stream Driver Architecture

VxWorks UDP Stream API Examples

The VxWorks UDP Stream API provides the application interface to send and receive streams of UDP datagrams. Examples of the stream functions available within this API are:

XgeStreamUdpSendSetup	- Set up a socket for UDP stream sends
XgeStreamUdpReceiveSetup	- Set up a socket for UDP stream receives
XgeStreamUdpSendMulti	- Perform a UDP stream send
XgeStreamUdpReceiveMulti	- Perform a UDP stream receive
XgeStreamSendClose	- Close a stream send socket
XgeStreamReceiveClose	- Close a stream receive socket

VxWorks Network/Stream Driver Version Support

The XGE driver currently supports VxWorks versions 6.x.

The driver is supplied in the form of a VxWorks object archive which is linked in to the user's bootable VxWorks image project. Like any network driver the XGE 10G driver is loaded into the system by making an entry in the BSP device table. Note that the XGE ports are available for network data transfer only after they are explicitly attached to the stack and configured with an appropriate network address and netmask.

XGE Windows Driver Support

Windows Network/RDMA Driver

The XGE Windows network driver provides two modes of operation. The network mode uses the standard Windows network stack, thus provides full compatibility with all Linux network applications, and any other applications that use the sockets API. The XGE hardware fully leverages the offload capabilities that are supported by Windows. The principal user interface for the network driver is the standards sockets API.

The RDMA mode provides for RDMA over Converged Ethernet (RoCE) operation. The principle user APIs for this mode are the IB Verbs API and the RDMA API. See the Linux RDMA API section earlier in this document for examples of user callable RDMA functions.

Windows Network/RDMA Driver Version Support

Supported Windows versions include Windows 7, Windows Server 2008, and Windows Server 2012